Guidelines/ Best practices/ Design patterns

When should two parts of the application be covered in a common function and when in separate ones?

Rule 1 - Changing number of elements

A diagram of a flowchart

Description automatically generated

|  |  |
| --- | --- |
| Merge | Split |
| One function produces one output, and another function consumes exactly that output | **The first function produces multiple outputs, and the second function needs to be called for each of them** |
| Keeping the functions separate would create overhead | **The execution of the second function needs to be parallelized to improve performance** |

Split functions when:

* **Large output array from A**: If function A generates a significant amount of data (e.g., more than 5 elements)
* **Complex or time-consuming processing in functions B**: If function B involves significant calculations, data manipulation, or external integrations.
* **Loosely coupled functionality**: If functions A and B operate on unrelated data or have independent logic with no dependency on each other's execution order or results (so the functions B can be called concurrently)

e.g.: function A query the DB and collect a specific list of users and for each user the system should send an email. The task to send the email for each user can be done concurrently in *n* number of B function.

Rule 2 - Alternative functions resulting from OR-nodes.

A diagram of a diagram

Description automatically generated

|  |  |
| --- | --- |
| Merge | Split |
| Functions A, B, C are together with a condition criterion. | **Functions B and C are separated and between them and the parent function (A) is a Choice (in a step function)** |
| Keeping the functions separate would create too much overhead | **Having for each branches a different function save resources, because function A doesn’t contain anymore logic which is not used in every call** |

Split functions when:

* **The conditional branches are large and complex**: If functions **B & C** are large/complex (take in consideration: lines of code, cyclomatic complexity and nesting depth)
* **Exists multiple branches:** if exists multiple branches (e.g., more than 3)
* **Function A doesn’t need to pass big amount of data to function B/C:** if parent function doesn’t pass data to its children’s or pass a small amount of data.
* **The condition is simple:** if the criteria condition is a simple evaluation.

e.g.: Imagine we have a serverless application for a file processing service. It receives files containing different data formats (e.g., CSV, JSON, XML) and needs to process them accordingly.

**Rule 3 -** Multiple uses

A diagram of a diagram

Description automatically generated

|  |  |
| --- | --- |
| Merge | Split |
| Function C is merged with A and with B | **The shared function, C, is extracted and called independently from A and from B** |
| Keeping the functions separate would create too much overhead | **The shared part easier to maintain** |

Split functions when:

* **Function C is part of several branches of the program flow:** if a function is used by multiple, independent parts (e.g., at least in 3 places => extract it separated: achieve only maintenance benefit without performance benefit)
* **Functions (A, B, C) are large and complex:** if having the shared function together with A/B exceed the total runtime limit or resource limit offered by the provider
* **Function A/B doesn’t need to pass big amount of data to function C:** if input functions (A/B) don’t pass data to its children’s or pass a small amount of data.

e.g.: Imagine we have an e-commerce application with the following functions:

* **ProcessOrder**: Receives and validates a new order, calculates shipping costs, and generates a unique order ID.
* **UpdateInventory**: Decreases stock levels for the ordered items in the inventory database.
* **SendOrderConfirmation**: Sends a personalized email confirmation to the customer with order details and estimated delivery time.
* **ProcessPayment**: Charges the customer's credit card for the order amount

Why Split:

* **UpdateInventory** and **SendOrderConfirmation** used independently: These functions are used in other parts of the application as well (e.g., order cancellation might trigger inventory update and email).
* **Small data passed**: **ProcessOrder** only needs to pass essential order data to each subsequent function, not the entire order object.

Rule 4 - Varying resource requirements

A diagram of a diagram

Description automatically generated

|  |  |
| --- | --- |
| Merge | Split |
| Function A and B together, allocate enough memory to handle the peak memory requirements of function B, even though function A does not need that much memory | **One function requires much less resources than another function** |
| Keeping the functions separate would create too much overhead | **For each function allocate only the necessary memory** |

Split functions when:

* **Resource requirements vary greatly between two functions:** if a function uses very high memory requirements, but the other not.
* **Functions A resources together with function B resources exceed the max resource limit offered by the cloud provider:** if keeping together overhead the max limit offered by the cloud provider.
* **Function A doesn’t need to pass big amount of data to function B.**

e.g.: Imagine we have a serverless application for image processing. It performs two main tasks:

* Resize images: Reduces the resolution of uploaded images to different sizes (e.g., thumbnails, previews).
* Apply watermarks: Adds a copyright watermark to the processed images.

Why Split:

* **Varying resource requirements**: Resizing images often requires more memory and CPU resources than watermarking, especially for large images.

Rule 5 - Long-running functions

A diagram of a flowchart

Description automatically generated

|  |  |
| --- | --- |
| Merge | Split |
| Function A and B together, if doesn’t exceed the run time limit offered by cloud provider | **The total run time could exceed the limit, then split them** |
| Keeping the functions separate would create too much overhead |  |

Split functions when:

* **The expected run time of a function is already close to the limit defined by the cloud provider:** the total run time could exceed the limit and lead to an abortion.

e.g.: Imagine we have a serverless application for data analysis. It reads large datasets from a storage service, performs complex calculations, and generates reports.

* **ReadData** (separate function): Reads the data from the storage service and stores it temporarily (e.g., in a cache or message queue).
* **AnalyzeData** (separate function): Retrieves the data from the temporary storage, performs the complex calculations, and stores the results (e.g., in another cache or queue).
* **GenerateReport** (separate function): Retrieves the analyzed data and generates the reports in different formats

Rule 6 - Too different functions

A diagram of a flowchart

Description automatically generated

|  |  |
| --- | --- |
| Merge | Split |
| When functions have similitudes. | **Two functions cannot be merged because they are too different** |

Split functions when:

* **Different concerns:** If the functions address entirely different aspects of the system.
* **Independent logic**: When functions have distinct decision-making logic or operate on unrelated data.
* **Functions are written in different languages:** two different programming languages with different requirements for the environment are used.

e.g.: Imagine you have a serverless application for a travel booking platform. It consists of two main functionalities:

* **Flight search**: Searches for flights based on user criteria (origin, destination, dates) and returns available options with prices.
* **Hotel booking**: Allows users to book hotels based on their chosen destination and dates, handling reservations and payments.

Why Split:

* **Different concerns**: Flight search and hotel booking address entirely different aspects of the booking process (finding flights vs. booking accommodations).
* **Potential language differences**: Depending on your development preferences, you might consider using different languages:

Split Implementation:

* **SearchFlights**: Written in Python or another language suitable for data analysis and manipulation. Takes user criteria, interacts with flight APIs, and returns available options.
* **BookHotel**: Written in Node.js or another language efficient for asynchronous tasks and payment processing. Takes user selection, interacts with hotel booking APIs, handles payments, and confirms reservations.